Introduction to Agentic AI

-- Context Engineering

Instructor: Guangjing Wang

guangjingwang@usf.edu

Last Lecture

- Instruction tuning
  - Chat Template
  - Full Parameter SFT
  - LoRa-based SFT
- Preference alignment
  - RLHF
  - DPO

This Lecture

- Context Engineering
- Prompt Design

LLM for Software Development

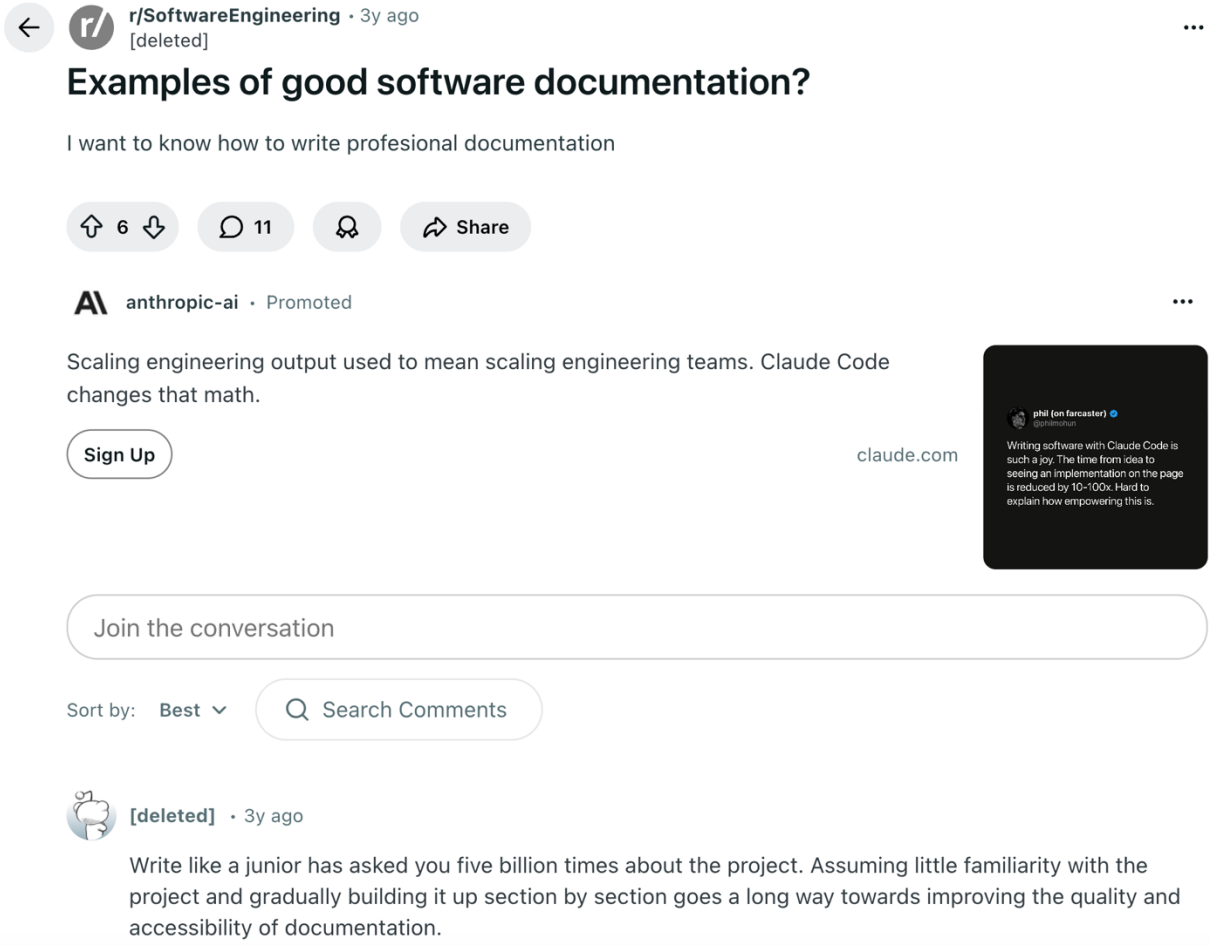
- Strengths
  - Expert-level code completion; Code understanding; Code fixing
- Limitations
  - Hallucinations
    - Generating non-existent/out-of-date APIs (mitigated with robust context engineering)
  - Context window limits
    - ~100K-100M tokens but not all are created equal
  - Latency
    - Seconds to minutes per request depending on task (plan and delegate accordingly)
  - Cost
    - $1-3 per million input tokens, $10+ per million output tokens for best models

Modern Software Developer

- The Structured Writing for Professionals:
  - Business Understanding
  - LLMs are only as good as you are
    - **Good Context** leads to good code;
    - If you cannot understand your code base, neither will an LLM;
    - There is no established software pattern yet for LLM;
    - Figure it out by yourself;
    - Read and review a lot of code;
- When using LLM:
  - Be explicit about what you want (languages, tech stacks, libraries, constraints)
  - Decompose tasks

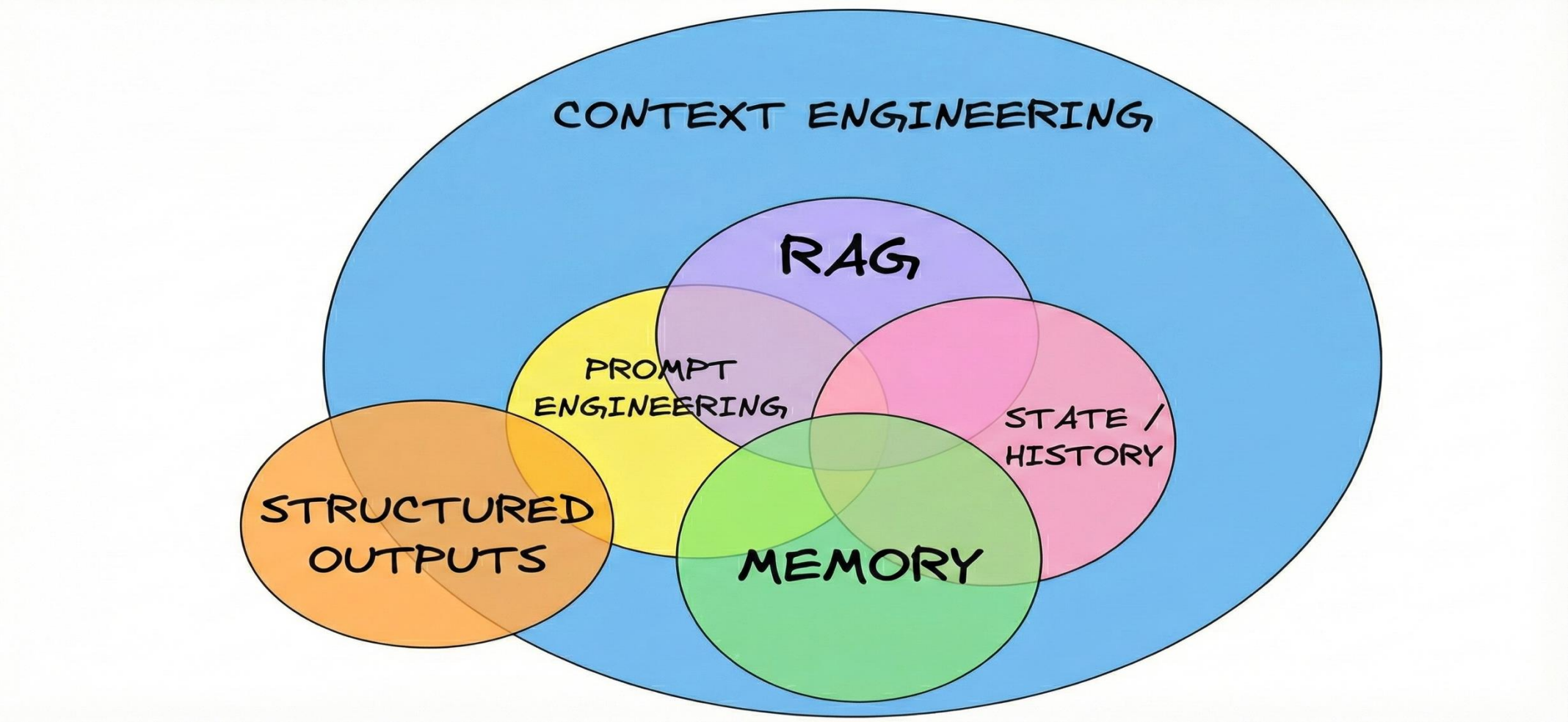
Prompting as Software Document

- Write like a junior has asked you five billion times about the project.
- Assuming little familiarity with the project and gradually building it up section by section goes a long way towards improving the quality and accessibility of documentation.



https://www.reddit.com/r/SoftwareEngineering/comments/y3ul0n/examples_of_good_software_documentation/

Context Engineering



https://x.com/dexhorthy/status/1933283008863482067

Context Engineering

- Designing and managing prompt chains (when applicable)
- Tuning instructions and system prompts
- Managing dynamic elements of the prompt (e.g., user inputs, date, etc.)
- Searching and preparing relevant knowledge (i.e., RAG)
- Query augmentation
- Tool definitions and instructions (in the case of agentic systems)
- Preparing and optimizing few-shot demonstrations
- Structuring inputs and outputs (e.g., delimiters, JSON schema)
- Short-term memory (i.e., managing state/historical context) and long-term memory (e.g., retrieving relevant knowledge from a vector store)
- And the many other tricks that are useful to optimize the LLM system prompt to achieve the desired tasks.

Context Engineering VS. Prompt Engineering

- Context Engineering:
  - The process of **tuning the instructions and relevant context** that an LLM needs to perform its tasks;
  - Iterative process to optimize instructions and the context you provide an LLM system to achieve a desired result.

- Prompt Engineering:
  - Designing the prompt about the context and structure.

Prompt Design

- **Specificity and Clarity:** clearly articulate the desired outcome;
- **Structured Inputs and Outputs:**
  - JSON, Markdown, XML; List, Paragraph, or Code Snippet;
- **Delimiters for Enhanced Structure:**
  - Utilizing special characters as delimiters within prompts
- **Task Decomposition for Complex Operations:**
  - breaking down complex processes into simpler subtasks;
  - Make the model to focus on each subtask individually;

An Example of Using Markdown and XML Tags

```
# Identity

You are coding assistant that helps enforce the use of snake case
variables in JavaScript code, and writing code that will run in
Internet Explorer version 6.

# Instructions

* When defining variables, use snake case names (e.g. my_variable)
  instead of camel case names (e.g. myVariable).
* To support old browsers, declare variables using the older
  "var" keyword.
* Do not give responses with Markdown formatting, just return
  the code as requested.

# Examples

<user_query>
How do I declare a string variable for a first name?
</user_query>

<assistant_response>
var first_name = "Anna";
</assistant_response>
```

ChatGPT 5.2 Prompting Guide (Example 1)

```
<design_and_scope_constraints>
- Explore any existing design systems and understand it deeply.
- Implement EXACTLY and ONLY what the user requests.
- No extra features, no added components, no UX embellishments.
- Style aligned to the design system at hand.
- Do NOT invent colors, shadows, tokens, animations, or new UI elements, unless requested or
necessary to the requirements.
- If any instruction is ambiguous, choose the simplest valid interpretation.
</design_and_scope_constraints>
```

https://developers.openai.com/cookbook/examples/gpt-5/gpt-5-2_prompting_guide/

ChatGPT 5.2 Prompting Guide (Example 2)

```
<extraction_spec>
You will extract structured data from tables/PDFs/emails into JSON.

- Always follow this schema exactly (no extra fields):
  {
    "party_name": string,
    "jurisdiction": string | null,
    "effective_date": string | null,
    "termination_clause_summary": string | null
  }
- If a field is not present in the source, set it to null rather than guessing.
- Before returning, quickly re-scan the source for any missed fields and correct omissions.
</extraction_spec>
```

https://developers.openai.com/cookbook/examples/gpt-5/gpt-5-2_prompting_guide/

ChatGPT 5.2 Prompting Guide (Example 3)

```
<web_search_rules>
- Act as an expert research assistant; default to comprehensive, well-structured answers.
- Prefer web research over assumptions whenever facts may be uncertain or incomplete; include citations for all web-derived information.
- Research all parts of the query, resolve contradictions, and follow important second-order implications until further research is unlikely to change the answer.
- Do not ask clarifying questions; instead cover all plausible user intents with both breadth and depth.
- Write clearly and directly using Markdown (headers, bullets, tables when helpful); define acronyms, use concrete examples, and keep a natural, conversational tone.
</web_search_rules>
```

https://developers.openai.com/cookbook/examples/gpt-5/gpt-5-2_prompting_guide/

Prompts for Agentic Tasks (Example 4)

Plan tasks thoroughly to ensure complete resolution

```
Remember, you are an agent - please keep going until the user's
query is completely resolved, before ending your turn and yielding
back to the user. Decompose the user's query into all required
sub-requests, and confirm that each is completed. Do not stop
after completing only part of the request. Only terminate your
turn when you are sure that the problem is solved. You must be
prepared to answer multiple queries and only finish the call once
the user has confirmed they're done.

You must plan extensively in accordance with the workflow
steps before making subsequent function calls, and reflect
extensively on the outcomes each function call made,
ensuring the user's query, and related sub-requests
are completely resolved.
```

https://platform.openai.com/docs/guides/prompt-engineering

Zero-shot VS. Few-shot Prompting

Write a for-loop iterating over a list of strings using the naming convention in our repo.

Write a for-loop iterating over a list of strings using the naming convention in our repo. Here are some examples of how we typically format variable names.

<example>
var StRaRrAy = ['cat', 'dog', 'wombat']
</example>

<example>
def func CaPiTaLiZeStR = () => {}
</example>

Multi-shot Chain-of-Thought

Write a function to check if a number is a perfect cube and a perfect square.

I want to write a function to check if a number is a perfect cube and a perfect square. Make sure to provide your reasoning first. Here are some examples of how to provide reasoning for a coding task.

<example>
Write a function that finds the maximum element in a list.
Steps: Initialize a variable with the first element. Traverse the list, comparing...
</example>

<example>
Write a function that checks is a number is a palindrome
Steps: Take the number. Reverse the elements in the numbers. Check if ...
</example>

Zero-Shot Chain-of-Thought

Write a function to find the longest increasing subsequence in an array.

Write a function to find the longest increasing subsequence in an array.

Think step by step about the subproblems before coding. Include worked out examples of subarrays you are considering as you answer this question.

Reducing Hallucination: Self-Consistency Prompting

What's the root cause for this error:
Traceback (most recent call last):
File "example.py", line 3, in <module>
print(nums[i])
IndexError: list index out of range

What's the root cause for this error:
Traceback (most recent call last):
File "example.py", line 3, in <module>
print(nums[i])
IndexError: list index out of range

Prompt 5x

Take majority result

Reducing Hallucination: Use Tools

After you have fixed this IndexError can you ensure that the CI tests still pass?

Fix the IndexError. Ensure the CI tests still pass once you have made the fix. Here are the available tools.

<tools>
pytest -s /path/to/unit_tests

pytest -v /path/to/integration_tests
</tools>

Reducing Hallucination: RAG

Extend the UserAuthService class to check that the client provides a valid OAuth token.

I want to extend the UserAuthService class to check that the client provides a valid OAuth token.

Here is how the UserAuthService works now:
<code_snippet>
def issue_oauth_token():
....
</code_snippet>

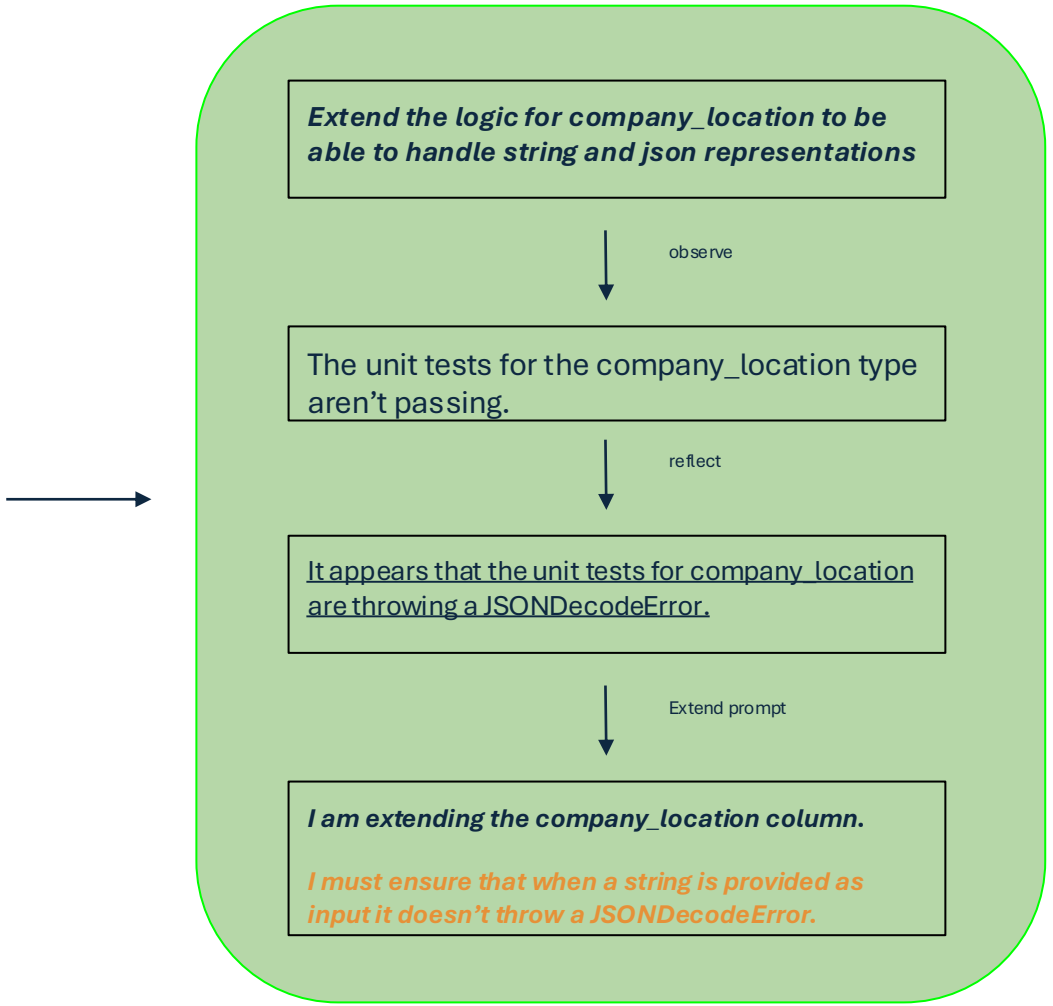
Here is the path to the requests-oauthlib documentation:
<url>
https://requests-oauthlib.readthedocs.io/en/latest/
</url>

Reducing Hallucination: Reflection

- Make the LLM *reflect* on its output
- Verbal feedback from environment signals are reincorporated into the LLM by augmenting the context
- After an action is taken in an environment and an observation made, add a prompt suffix:
  - “Now critique your answer. Was it correct? If not, explain why and try again.”
- Multi-turn prompting
  - Turn 1: Model gives a first try.
  - Turn 2: You ask “*Was that correct? Reflect and revise if needed*.”

Reflection

Ensure that the company_location column can handle string and json representations.

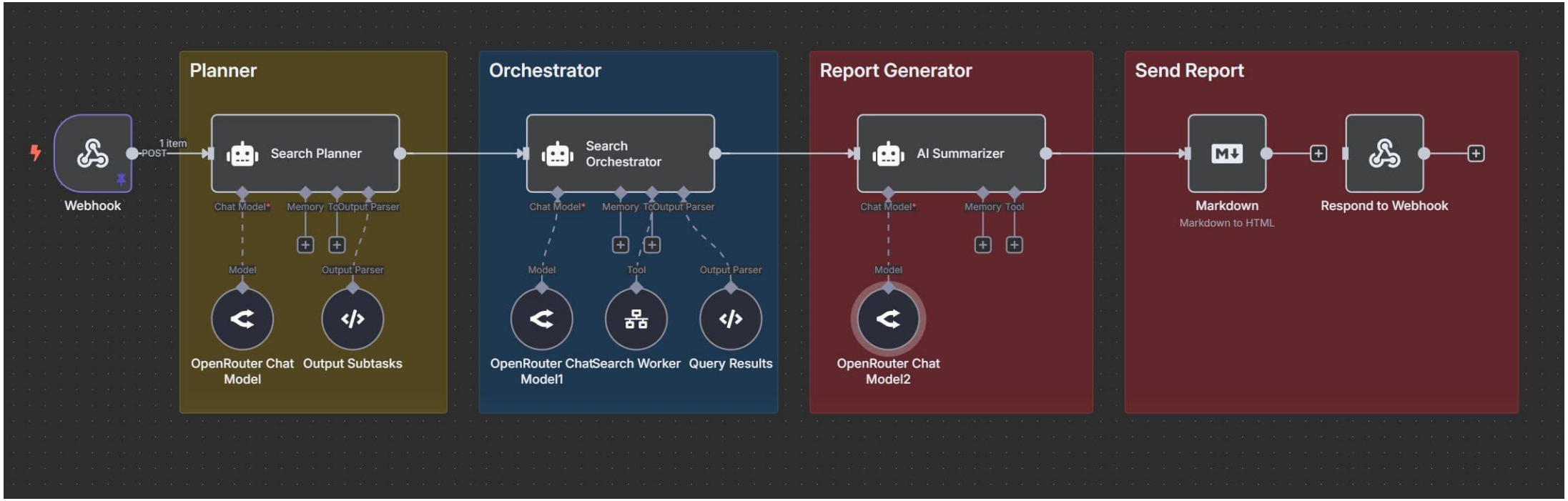


Why Prompt Design Matters and Works

- Context and Prompt Engineering is Art and Science
- Prompts function as selectors, extracting specific task-relevant information from the model's full hidden state during CoT reasoning
- Each prompt defines a unique trajectory through the answer space, and the choice of this trajectory is crucial for task performance and future navigation in the answer space.

Why Prompt Design Matters and Works: A Complexity Analysis of Prompt Search Space in LLMs (ACL'2025) https://aclanthology.org/2025.acl-long.1562/

Example: An Agentic Workflow



Basic Agentic Layers

- **System Layer**: Core agent identity and capabilities;
- **Task Layer**: Specific instructions for the current task;
- **Tool Layer**: Descriptions and usage guidelines for each tool;
- **Memory Layer**: Relevant historical context and learnings;

System Prompt for Search Planner Agent

```
You are an expert research planner. Your task is to break down a complex research query (delimited
by <user_query></user_query>) into specific search subtasks, each focusing on a different aspect
or source type.

The current date and time is: {{ $now.toISO() }}
For each subtask, provide:
1. A unique string ID for the subtask (e.g., 'subtask_1', 'news_update')
2. A specific search query that focuses on one aspect of the main query
3. The source type to search (web, news, academic, specialized)
4. Time period relevance (today, last week, recent, past_year, all_time)
5. Domain focus if applicable (technology, science, health, etc.)
6. Priority level (1-highest to 5-lowest)

All fields (id, query, source_type, time_period, domain_focus, priority) are required for each
subtask, except time_period and domain_focus which can be null if not applicable.

Create 2 subtasks that together will provide comprehensive coverage of the topic. Focus on
different aspects, perspectives, or sources of information.
Each substask will include the following information:
id: str
query: str
source_type: str  # e.g., "web", "news", "academic", "specialized"
time_period: Optional[str] = None  # e.g., "today", "last week", "recent", "past_year", "all_time"
domain_focus: Optional[str] = None  # e.g., "technology", "science", "health"
priority: int  # 1 (highest) to 5 (lowest)
After obtaining the above subtasks information, you will add two extra fields. Those correspond to
start_date and end_date. Infer this information given the current date and the time_period
selected. start_date and end_date should use the format as in the example below:
"start_date": "2024-06-03T06:00:00.000Z",
"end_date": "2024-06-11T05:59:59.999Z",
```

Instructions

This prompt is somewhat abstract:

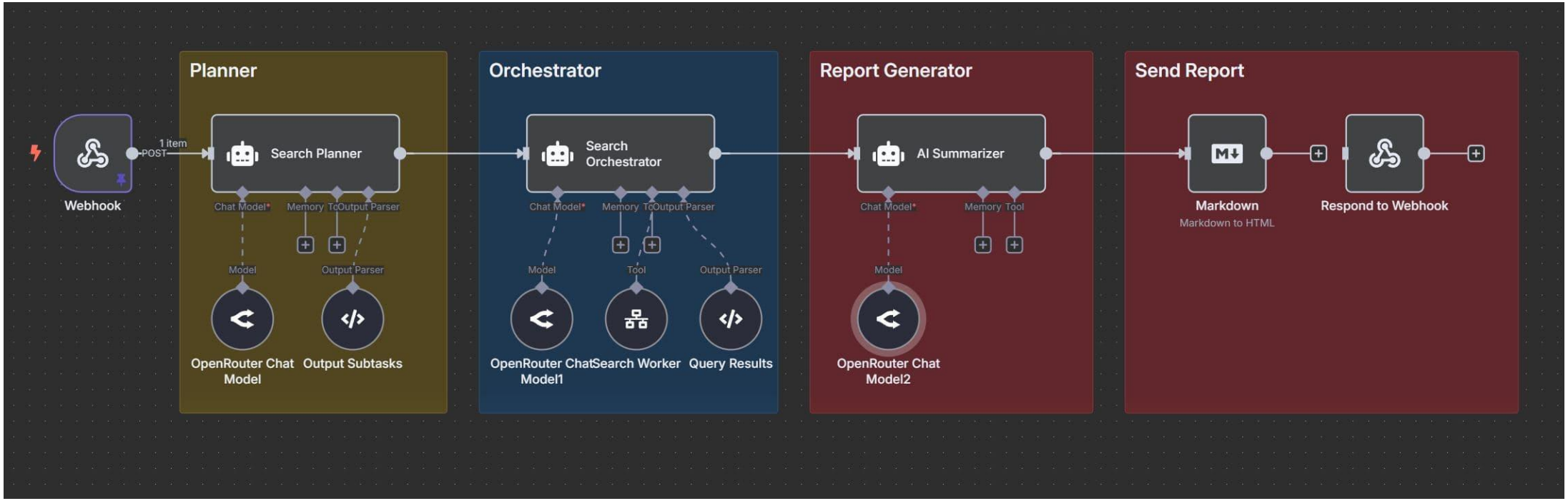
```
You are an expert research planner. Your task is to break down a complex research query (delimited by <user_query></user_query>) into specific search subtasks, each focusing on a different aspect or source type.
```

This prompt is more detailed:

```
You are an expert research planner. Your task is to break down a complex research query (delimited by <user_query></user_query>) into specific search subtasks, each focusing on a different aspect or source type.

The current date and time is: {{ $now.toISO() }}
For each subtask, provide:
1. A unique string ID for the subtask (e.g., 'subtask_1', 'news_update')
2. A specific search query that focuses on one aspect of the main query
3. The source type to search (web, news, academic, specialized)
4. Time period relevance (today, last week, recent, past_year, all_time)
5. Domain focus if applicable (technology, science, health, etc.)
6. Priority level (1=highest to 5=lowest)

All fields (id, query, source_type, time_period, domain_focus, priority) are required for each subtask, except time_period and domain_focus which can be null if not applicable.

Create 2 subtasks that together will provide comprehensive coverage of the topic. Focus on different aspects, perspectives, or sources of information.
Each substask will include the following information:
id: str
query: str
source_type: str  # e.g., "web", "news", "academic", "specialized"
time_period: Optional[str] = None  # e.g., "today", "last week", "recent", "past_year", "all_time"
domain_focus: Optional[str] = None  # e.g., "technology", "science", "health"
priority: int  # 1 (highest) to 5 (lowest)
After obtaining the above subtasks information, you will add two extra fields. Those correspond to start_date and end_date. Infer this information given the current date and the time_period selected. start_date and end_date should use the format as in the example below:
"start_date": "2024-06-03T06:00:00.000Z",
"end_date": "2024-06-11T05:59:59.999Z",
```

- Providing more context that we need to give the system to work as we want;
- It informs the system more about the problem scope and the specifics of what exactly we desire from it.

Question 1: Incomplete Task Execution



- **Problem**: When running the agentic workflow, the orchestrator agent often creates three search tasks but only executes searches for two of them, skipping the third task without explicit justification.

Question 1: Incomplete Task Execution

- **Solution**: Two approaches are possible:
  - **Flexible Approach**: Allow the agent to decide which searches are necessary, but require explicit reasoning for skipped tasks
  - **Strict Approach**: Add explicit constraints requiring search execution for all planned tasks

```
You are a deep research agent responsible for executing comprehensive research tasks.
TASK EXECUTION RULES:
- For each search task you create, you MUST either:
  1. Execute a web search and document findings, OR
  2. Explicitly state why the search is unnecessary and mark it as completed with justification
- Do NOT skip tasks silently or make assumptions about task redundancy
- If you determine tasks overlap, consolidate them BEFORE execution
- Update task status in the spreadsheet after each action
```

Question 2: Lack of Debugging Visibility

- **Problem**: Without proper logging and state tracking, it was difficult to understand why the agent made certain decisions.
- **Solution**: For this example, we can implement a task management system using a spreadsheet or text file (for simplicity) with the following fields:
  - Task ID
  - Search query
  - Status (todo, in_progress, completed)
  - Results summary
  - Timestamp

Question 3: Ignoring Error Cases

- **Problem**: Context does not specify behavior when things go wrong.
- **Solution**: In some cases, it helps to add error handling instructions to your AI Agents:

```
ERROR HANDLING:
- If a search fails, retry once with a rephrased query
- If retry fails, document the failure and continue with remaining tasks
- If more than 50% of searches fail, alert the user and request guidance
- Never stop execution completely without user notification
```

Make Expectations Explicit

- Don't assume the agent knows what you want. Be explicit about:
  - Required vs. optional actions
  - Quality standards
  - Output formats
  - Decision-making criteria
- Build debugging mechanisms into your agentic system:
  - Log all agent decisions and reasoning
  - Track state changes in external storage
  - Record tool calls and their outcomes
  - Capture errors and edge cases

Eliminate Prompt Ambiguity

Bad Example:

```
Perform research on the given topic.
```

Good Example:

```
Perform research on the given topic by:
1. Breaking down the query into 3-5 specific search subtasks
2. Executing a web search for EACH subtask using the search_tool
3. Documenting findings for each search in the task tracker
4. Synthesizing all findings into a comprehensive report
```

https://www.promptingguide.ai/agents/context-engineering

Structured Input and Output

- Effort on the details related to the subtasks the planning agent needs to produce
  - Structure a list of the required information
  - Hints and examples

```
For each subtask, provide:
1. A unique string ID for the subtask (e.g., 'subtask_1', 'news_update')
2. A specific search query that focuses on one aspect of the main query
3. The source type to search (web, news, academic, specialized)
4. Time period relevance (today, last week, recent, past_year, all_time)
5. Domain focus if applicable (technology, science, health, etc.)
6. Priority level (1-highest to 5-lowest)

All fields (id, query, source_type, time_period, domain_focus, priority) are required for each
subtask, except time_period and domain_focus which can be null if not applicable.

Create 2 subtasks that together will provide comprehensive coverage of the topic. Focus on
different aspects, perspectives, or sources of information.
```

Output Format Specification

Providing some context on the subtask format and field types that we expect

```
Each substask will include the following information:
id: str
query: str
source_type: str  # e.g., "web", "news", "academic", "specialized"
time_period: Optional[str] = None  # e.g., "today", "last week", "recent", "past_year", "all_time"
domain_focus: Optional[str] = None  # e.g., "technology", "science", "health"
priority: int  # 1 (highest) to 5 (lowest)
```

The Reality of Context Engineering

- Building effective AI agents requires substantial tuning of system prompts and tool definitions.
- Hours iterating on:
  - System prompt design and refinement
  - Tool definitions and usage instructions
  - Agent architecture and communication patterns
  - Input/output specifications between agents
- Key factors: Task complexity, Available resources, Previous execution history, Error patterns

Measuring Context Engineering Success

- Track these metrics to evaluate context engineering effectiveness:
  - **Task Completion Rate**: Percentage of tasks completed successfully
  - **Behavioral Consistency**: Similarity of agent behavior across similar inputs
  - **Error Rate**: Frequency of failures and unexpected behaviors
  - **User Satisfaction**: Quality and usefulness of outputs
  - **Debugging Time**: Time required to identify and fix issues

- Treat context engineering as a multi-round activity:
  - **Systematic observation** of agent behavior
  - **Careful analysis** of failures and edge cases
  - **Iterative refinement** of instructions and constraints
  - **Rigorous testing** of changes

Again: Iterating (Optimizing) Based on Behavior

- Context engineering is an iterative process:
  - **Deploy** the agent with initial context
  - **Observe** actual behavior in production
  - **Identify** deviations from expected behavior
  - **Refine** system prompts and constraints
  - **Test** and validate improvements
  - **Repeat**
- Balance Flexibility and Constraints
  - **Strict constraints**: More predictable but less adaptable
  - **Flexible guidelines**: More adaptable but potentially inconsistent

Beyond Prompt: RAG and Memory

- RAG: Retrieval Augmented Generation
- If a similar query was already used by a user before, it is possible to store those results in a vector store and search over them to avoid the need to create a new set of subqueries for a plan that we already generated and exists in the vector store.
- Context engineering is not just about optimizing your prompt; it’s about choosing the right context for the goals you are targeting.
- You can also get more creative about how you are maintaining that vector store and how you pull those existing subtasks into context.

Other Context Engineering Topics

- Context compression
- Context management techniques
- Context safety
- Evaluating context effectiveness
- Automating the process of Context Engineering

References

- https://themodernsoftware.dev/
- https://www.promptingguide.ai/guides/context-engineering-guide